Bachelor of Science in Electronic and Telecommunication Engineering

Improvement of Security for IoT Based Network Using MQTT

Submitted by

MD. Akibur Rahman

T-153005

Shihab Uddin

T-153032

Supervised by:

Engr. Md. Jiabul Hoque

Lecturer

Department of ETE

International Islamic University Chittagong

Department of Electronic and Telecommunication Engineering

International Islamic University Chittagong

Kumira, Sitakunda, Chattogram - 4318

Academic Year: 2020

DECLARATION

It is hereby declared that this work has been done by us and no portion of the work contained in this thesis/project has been submitted elsewhere for the award of any degree or diploma.

_____

MD. Akibur Rahman

_____

Shihab Uddin

DEDICATION

This thesis work is dedicated to all of our honourable teachers and parents.

CERTIFICATE OF APPROVAL

This is to certify that **MD. Akibur Rahman** (T-153005) and **Shihab Uddin** (T-153032), are the students of the department of Electronic and Telecommunication Engineering, International Islamic University Chittagong had carried out the Thesis on **"Improvement of Security for IoT Based Network Using MQTT''** successfully under my supervision and guidance.

_____

Supervisor
Engr. Md. Jiabul Hoque
Lecturer,
Department of Electronic and Telecommunication Engineering
International Islamic University Chittagong.

ACKNOWLEDGMENT

In the name of Allah, the Entirely Merciful, the Especially Merciful. All praises and glory are to Allah (SWT) for blessing us with opportunities abound and showering upon us His mercy and guidance all through the life. And may peace and blessings of Allah be upon Prophet Muhammad (pbuh), a guidance and inspiration to our lives. We express our sincere gratitude and indebtedness to our thesis supervisor **Engr. Md. Jiabul Hoque**, for his initiative in this field of research, valuable guidance, encouragement throughout the time of research. We also express our thankfulness to him for providing us with best facilities in the department and his timely suggestions. We are also thankful to **Engr. Syed Zahidur Rashid**, Convener of our thesis for his dedication and sacrifice. We also express our sincere gratitude to all our teacher for giving us the best effort throughout our entire academic years. And we remember our parents for their support in our life till now. And we also express gratitude our friends, well-wishers for their directly or indirectly involvement in the completion of this thesis work.

Authors

ABSTRACT

Rapid technological advances in the past made possible the miniaturization of network devices to meet the cost and power consumption requirements in IoT and M2M scenarios. What is missing in this picture is a radio technology with both long-range capability and a very low-cost footprint. Existing radio technologies such as 3G/4G or Short-Range Radio do not aptly meet the requirements of IoT scenarios because they are either too expensive or are not able to provide the required range. Other wireless technologies are geared towards high bandwidth, which is in most cases not a requirement for IoT. Emerging LPWAN technologies such as ETSI LTN or Lora WAN are poised for filling the gap by providing long range (up to 40km) and low power connectivity. These technologies allow low cost radio devices and operation thus enabling scaling up IoT applications

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Introduction

Messaging Queue Telemetry Transport is a Client Server publish/subscribe messaging transport protocol. Structured by IBM, it is initially expected for problematic systems with limited assets, for example, low transfer speed and high-inactivity. It comprises of one specialist server and two sorts of clients called as Publisher (Publish clients) and Subscriber (Subscribe clients). Broker server goes about as a middle person for messages sent between Publish clients and Subscribe clients for the fascinating theme. At the point when the Publish clients gives a subject and makes an impression on the Broker server, the Subscribe clients chooses the themes that it finds interesting [1]. It is open, simple, lightweight and designed to be easy to implement. These features make it ideal for use in many situations, including constrained environments such as for communication in Machine-to-Machine (M2M) and Internet of Things (IoT) contexts where a small code track is required and/or network bandwidth is at an exceptional. To ensure the reliability of messaging, MQTT supports three Levels of Quality of Services (QoS) [1].

The IoT exploits the essential technologies of these objects to transform it from conventional ones to astute. For example, Internet protocols, applications, embedded devices and communication technologies. In this manner, it is normal that the IoT is to contribute in developing of the world's economy and in improving the personal satisfaction.

Another name for IoT is the Internet of Objects, which comprises of a remote correspondence and a self-arranging network between objects. IoT is a stage for gadgets to discuss electronically with the world. It became as a portrayal for the quantity of advancements and research disciplines that permit the Internet to speak with physical items in reality.

In 1999, Andy Stanford-Clark of IBM and Arlen Nipper was introduced a protocol for messaging named MQTT (Message Queue Telemetry Transport). In 2013, the MQTT became standard protocol of the Organization for the Advancement of Structured Information Standards (OASIS) [2]. MQTT protocol connects the networks and devices with middleware and applications. This connection uses machine-to-server (M2S), server-to-server (S2S), machine-to-machine communication patterns, and routing mechanism (one-to-many, one-to-one, many-to-many) [3].

The default MQTT port that worked on is TCP/IP port.1883. MQTT has various sorts, for example, mosquito, hivemq, and paho MQTT. Transport Layer Security (TLS)/Secure Sockets Layer (SSL) are security conventions that give interchanges security through the PC arrange that is utilized in various applications, for example, email, web perusing, Internet faxing, voice-over-IP (VoIP), and texting.

This paper speaks to the Message Queuing Telemetry Transport (MQTT) protocol that has comprehensively used. Recently, renowned enterprises have utilized it. For example, Amazon and Facebook. MQTT speaks to the M2M conventions, that it depends on distribute/buy in correspondence design. The motivation behind this convention needs to utilize it in gadgets with restricted memory abilities and constrained handling power.

1.2 Motivation

Inspired by the abovementioned, this work presents a protected sending of MQTT on Wireless Sensor Nodes, emulating an IoT organization, assessing the presentation of different security choices on a genuine world testbed. In addition, the consequences of this assessment are surveyed with regards to a genuine modern system, featuring the most suitable choices with regards to a genuine mechanical application. Said evaluation depends on a follow investigation directed in a working breeze park's system (in Brande, Denmark); an agent use instance of mechanical systems, concentrated with regards to the European venture VirtuWind [4].

Internet of Things (IoT) is by and by a hot innovation around the world. The government, the scholarly world, and industry are associated with various parts of research, execution, and business with IoT. IoT cuts across various application area verticals going from non-military personnel to protection segments. These areas incorporate farming, space, social insurance, fabricating, development, water, and mining, which are directly progressing their inheritance framework to help IoT. Today it is conceivable to imagine unavoidable network, stockpiling, and calculation, which, thus, offers ascend to building distinctive IoT arrangements. IoT-based applications, for example, creative shopping framework, foundation the executives in both urban and country regions, remote wellbeing checking and crisis warning frameworks, and transportation frameworks, are steadily depending on IoT based frameworks. Hence, it is essential to gain proficiency with the basics of this rising innovation.

As the web keeps on developing, it has gotten in excess of a straightforward system of PCs, but instead a system of different gadgets, while IoT fills in as a system of different "associated" gadgets a system of systems. These days, gadgets like cell phones, vehicles, mechanical frameworks, cameras, toys, structures, home machines, modern frameworks and innumerable others would all be able to share data over the Internet.

Not with standing their sizes and capacities, these gadgets can achieve well-informed redesigns, following, situating, control, continuous checking and procedure control. In the previous years, there has been a significant engendering of Internet proficient gadgets. Despite the fact that its most critical business impact has been seen in the purchaser hardware field; for example especially the transformation of cell phones and the enthusiasm for wearable gadgets (watches, headsets, and so forth.), interfacing individuals has become simply a section of a greater development towards the relationship of the computerized and physical universes [6].

1.3 Objectives

The main objectives of this project have pointed below.

- Develop a MQTT broker.
- Design and developed a secure MQTT protocol for communication.
- Develop faster medium for device communication.
- Ensure the no data loss in communication.

1.4 Overview of the Report

Six section has covered in the course of design and development of this project. The sections and their substance are as per the following:

- Chapter 1 is the introductory chapter that gives the overview, motivation and objective of the project.

- Chapter 2 is Review of Previous Works. Previous work related of this project has discussed in this chapter.

- Chapter 3 is methodology of this system. In this chapter, all the methods used in this project has described elaborately.

- Chapter 4 compacts with the result and analysis of the project. In this chapter Objective, verification and system specification of the project has discussed.

- Finally, the summary of this project has discussed in detail in chapter 5. The limitation of the project, advantage and future development has discussed on this topic.

1.5 Summary

In this Chapter, we have discussed about the aims, motivation, objectives of the thesis, situation of the current system, necessity and brief overview of the proposed system.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

Internet of Things (IoT) has presently a hot innovation around the world. The government, the scholarly world, and industry are engaged with various parts of research, usage, and business with IoT. IoT cuts across various application area verticals going from regular citizen to guard areas. These areas incorporate horticulture, space, medicinal services, producing, development, water, and mining, which are directly changing their inheritance foundation to help IoT. Today it has conceivable to imagine inescapable network, stockpiling, and calculation, which, thus, offers ascend to building diverse IoT arrangements. IoT-based applications, for example, imaginative shopping framework, foundation the board in both urban and provincial territories, remote wellbeing checking and crisis notice frameworks, and transportation frameworks, are bit by bit depending on IoT based frameworks.

2.2 Review of Previous Works

2.2.1 Development of Message Queuing Telemetry Transport (MQTT) based Vehicle Accident Notification System.

Bharat Naresh Bansal, Vivek Garg.” Development of Message Queuing Telemetry Transport (MQTT) based Vehicle Accident Notification System.” International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-9 Issue-2, December, 2019. The model in this paper has an accident notification system ESP8266 Node MCU and a basic vibration sensor is the core of this framework. The vibration sensor continuously senses the vibrations and on exceeding a predefined threshold limit, sends out a notification to enrolled numbers. The traffic and the hurrying around on streets is relentless and with it the number of mishaps and street losses have massively expanded. Nevertheless, there has no simple and functional approach to decrease the use

of the autos. Consistently the humankind read about a huge number of individuals kicking the bucket of street losses and the vast majority of them pass on in light of the fact that the relations or the concerned ones of the reveled individuals are not convenient educated. The passing losses can be limited, as it were, by simply opportune advising the families of the concerned ones.

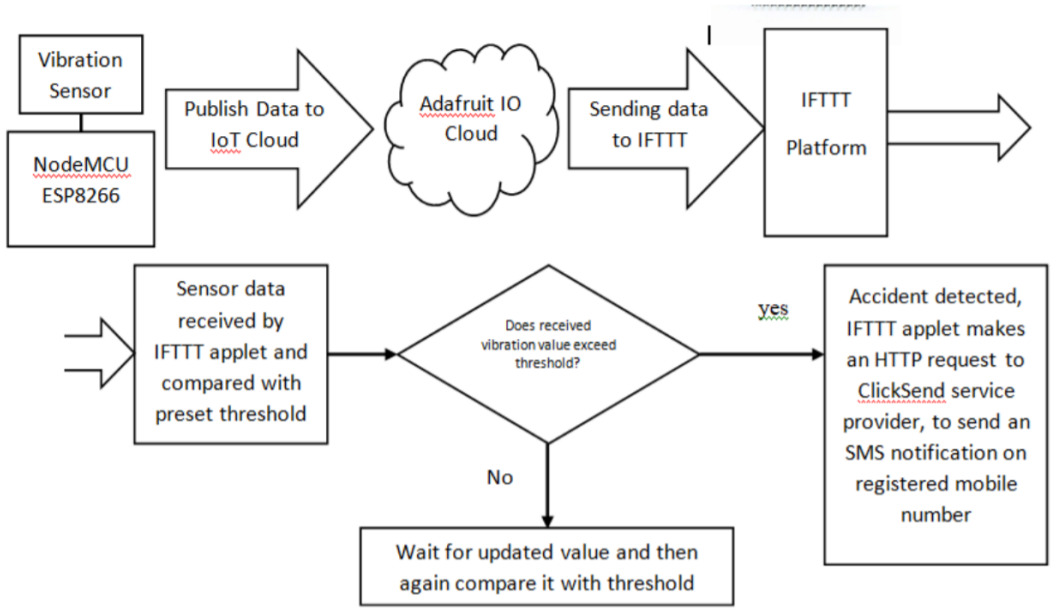


Figure 2.1: Diagram of the system [7].

The vibration sensor has interfaced to the Node MCU microcontroller. which ceaselessly peruses the sensor information and distributes it to the Adafruit IO stage utilizing MQTT convention and the IoT stage consistently sends the information to IFTTT, which has bought in to get the sensor information, the applet made on the IFTTT contrasts the got sensor esteem and preset limit, and if the worth surpasses edge, IFTTT makes a http solicitation to Click-Send to send a SMS notice on enrolled portable numbers [7].

2.2.2 Secure MQTT for Internet of Things (IoT).

Meena Singh, Rajan MA, Shivraj VL, and Balamuralidhar P, “Secure MQTT for Internet of Things (IoT)”. 2015 Fifth International Conference on Communication Systems and Network Technologies. In this paper addresses the concerns in the deployment of IoT to

ensure the security of devices and Device-to-Device communications. In this paper proposes a secure version of MQTT and MQTT-SN protocols in which security feature is augmented to the existing MQTT protocol based on Key/Cipher text Policy-Attribute Based Encryption (KP/CP-ABE) using lightweight Elliptic Curve Cryptography. they also demonstrate how feasible SMQTT and

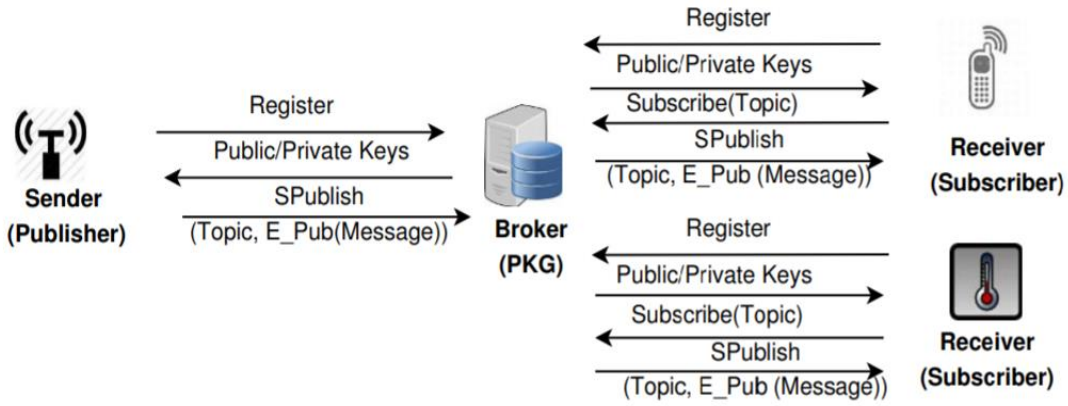


Figure 2.2: Architecture of the scheme [8].

SMQTT-SN are for various IoT. The implementations show that the paper still has issues related to security aspects of SMQTT such as key revocation, group publish/subscribe for distributed SMQTT. In this paper, they MQTT convention with KPABE and CP-ABE conspire referenced in dependent on lightweight ECC. To empower ABE, arrangement, encryption also, decoding activities should be performed. During arrangement stage, Broker (confided in outsider) renders the capacity of a Open Key Generator (PKG) produces the ace mystery key what's more, get to strategy. It distributes the open parameters for each property of the entrance strategy. Each gadget must enroll itself with an Identity and set of qualities with the PKG. PKG confirms traits and different subtleties given by the gadget. It sends open parameters to sending gadget to scramble the information. The private key for each credit is sent to recipient by PKG [8].

2.2.3 Correlation Analysis of MQTT Loss and Delay According to QoS Level.

S. Lee, H. Kim, D. K. Hong, and H. Ju, “Correlation analysis of MQTT loss and delay according to QoS level,” in International Conference on Information Networking, 2013,

pp. 714–717. The author analyzes MQTT message transmission process, which consists of real, wired/wireless publish client, broker server and Subscribe client. By the methodology, MQTT transmits messages

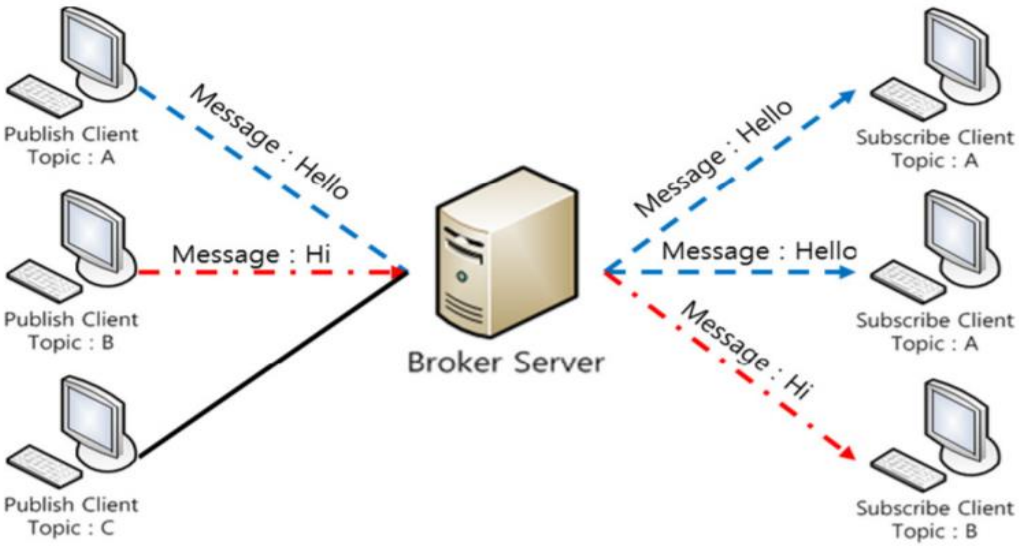


Figure 2.3: MQTT Message Transmission Process [9].

Through three levels of QoS with various sizes of payloads, it then captures packets to analyze end-to-end delays and message loss. Although the results suggested that end-to-end delay is significantly associated with message loss under different payloads, the behavior under various wQoS levels and payloads is still unclear [9].

2.2.4 Performance Evaluation of MQTT and CoAP via a Common Middleware.

D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing 2014. The aims to design and implement a common middleware that supports MQTT and CoAP and provides a common programming interface. For wireless sensor

networks to transfer data from a gateway to clients the paper uses the “publish-subscribe” architecture where a client needing data registers its interests with a server.

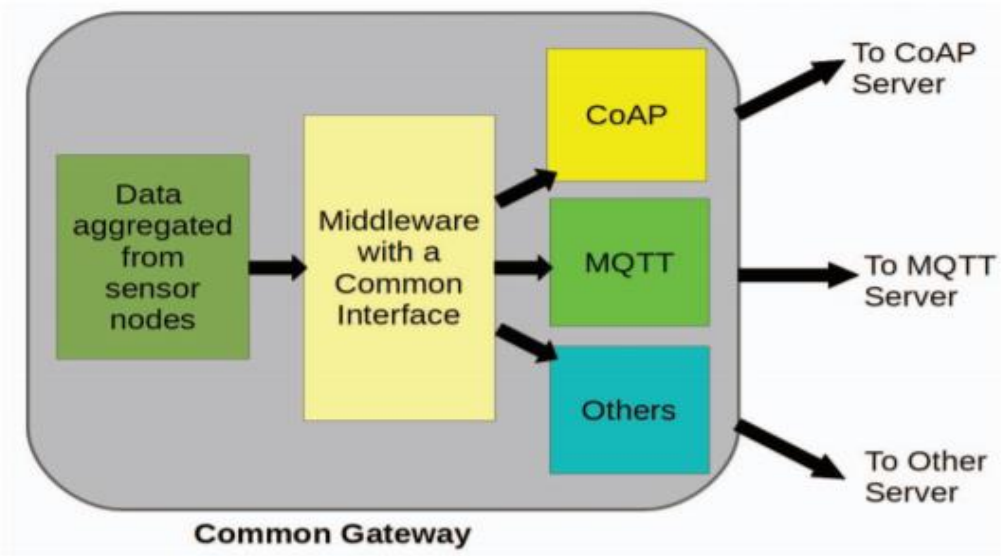


Figure 2.4: Design of the Middleware [10].

WSNs require bandwidth-efficient and energy-efficient application protocols for data transmission. Message Queue Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) are two such protocols proposed for resource-constrained devices. During this paper, design and implement a standard middleware that supports MQTT and CoAP and provides a typical programming interface. The middleware to be extensible to support future protocols. Using the common middleware, they conducted experiments to check the performance of MQTT and CoAP in terms of end-to-end delay and bandwidth consumption. Experimental results reveal that MQTT messages have lower delay than CoAP messages at lower packet loss rates and better delay than CoAP messages at higher loss rates.

2.2.5 Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things.

Y. F. Gomes, D. F. S. Santos, H. O. Almeida, and A. Perkusich, “Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things,” in *2015 IEEE*

International Conference on Consumer Electronics, ICCE 2015, 2015, pp. 200–201. This paper compares MQTT and CUPUS in the context of smart city application scenarios. Exhibited an associated wellbeing engineering that incorporates the ISO/IEEE 11073 detail and the MQTT convention. Other than the favorable circumstances for obliged assets gadgets, MQTT distribute/buy in correspondence model brings other alluring highlights, for example, the programmed revelation of gadgets by the utilization of Brokers. In the associated wellbeing setting, the proposed design displays another arrangement of situations, where MQTT specialists can be utilized to share wellbeing data in home systems and with the Internet.

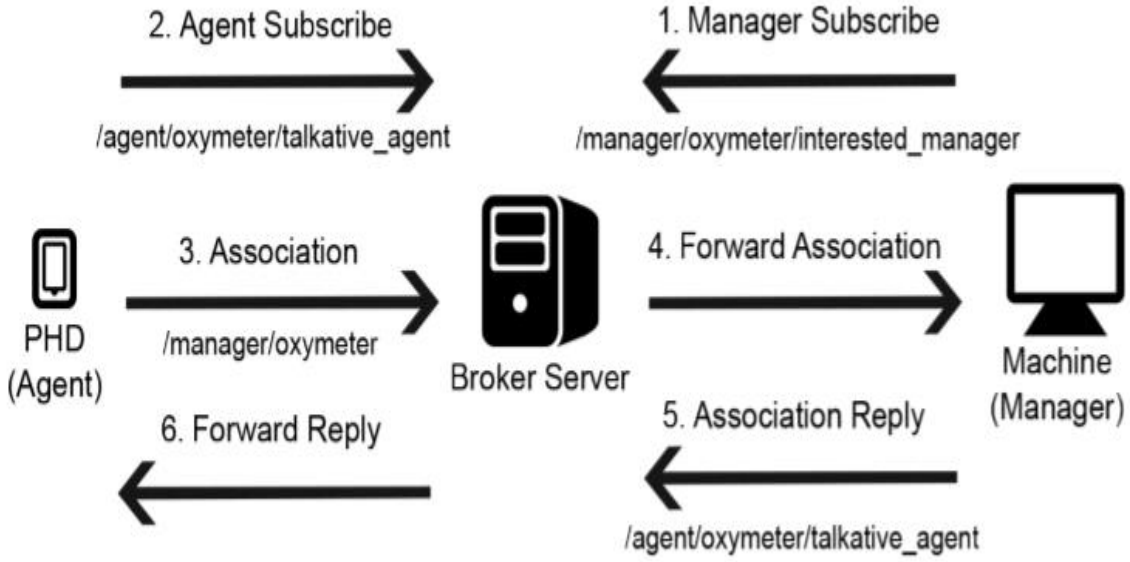


Figure 2.5: Device discover data flow of the system [11].

As future work, security highlights of MQTT in a wellbeing setting will be assessed, and furthermore the coordination of MQTT systems with other IoT conventions, for example, CoAP, utilizing ISO/IEEE 11073 as base convention ought to be contemplated. misfortune rates. The MQTT protocol proves to be suitable for Wireless Sensor Networks (WSNs) and heterogeneous environments due to its small code footprint, low bandwidth usage and standardized interfaces [11].

2.2.6 Experimental and Environmental Control and Monitoring of the Chemistry Laboratory.

S. Wilson and J. Frey, "The Smart Lab: Experimental and environmental control and monitoring of the chemistry Laboratory," in 2009 International Symposium on Collaborative Technologies and Systems, CTS 2009, 2009, pp. 85–90. This paper reconnoiters the use of the MQTT (Message Queue Telemetry Transport) lightweight protocol together with the ISO/IEEE 11073 standard carrying the IoT paradigm to the healthcare area where users are able to share Personal Health Information with their doctors and be able to use healthcare services using Personal Health Devices CE devices and the Internet.

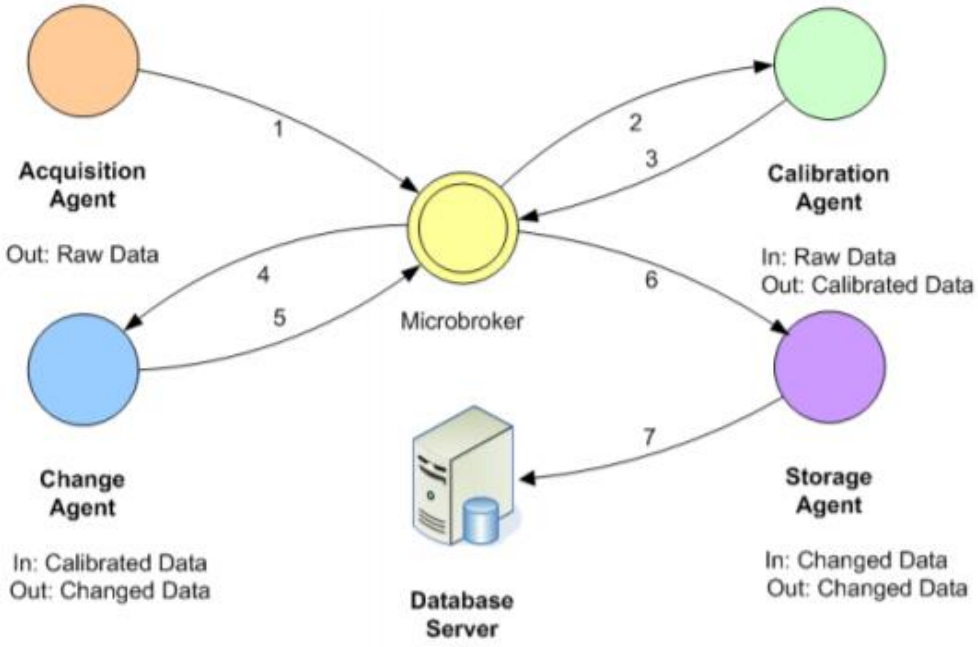


Figure 2.6: The core message pathway for environmental and experimental data [12].

The decay of the framework considered quicker coding and the executives of the framework and gave the capacity to deploy a distributed system. For instance, time-basic segments can run on dedicated equipment with a worldwide alignment framework running somewhere else. In spite of the fact that this framework was picked for information procurement from the Labjack UE9, the procedure could be altered for

various information sources such as savvy sensors; right now, securing and adjustment could be consolidated into a solitary segment. Variety observing can be by-passed when full information catch is required, for example, in trial checking. The paper present new ways of connecting PHDs in home networks and the Internet by the use of MQTT Brokers, which will reduce the amount of data traffic. The paper does not consider the security features of MQTT in a health context [12].

2.2.7 Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol.

S. Katsikeas *et al.*, "Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol," in *Proceedings - IEEE Symposium on Computers and Communications*, 2017, vol. 0, pp. 1193–1200. In This paper features the Message Queue Telemetry Transport (MQTT) as a lightweight convention appropriate for the industrial space, introducing a thorough assessment of various security components that could be utilized to secure the MQTT-empowered communications on a genuine testbed of remote sensor bits.

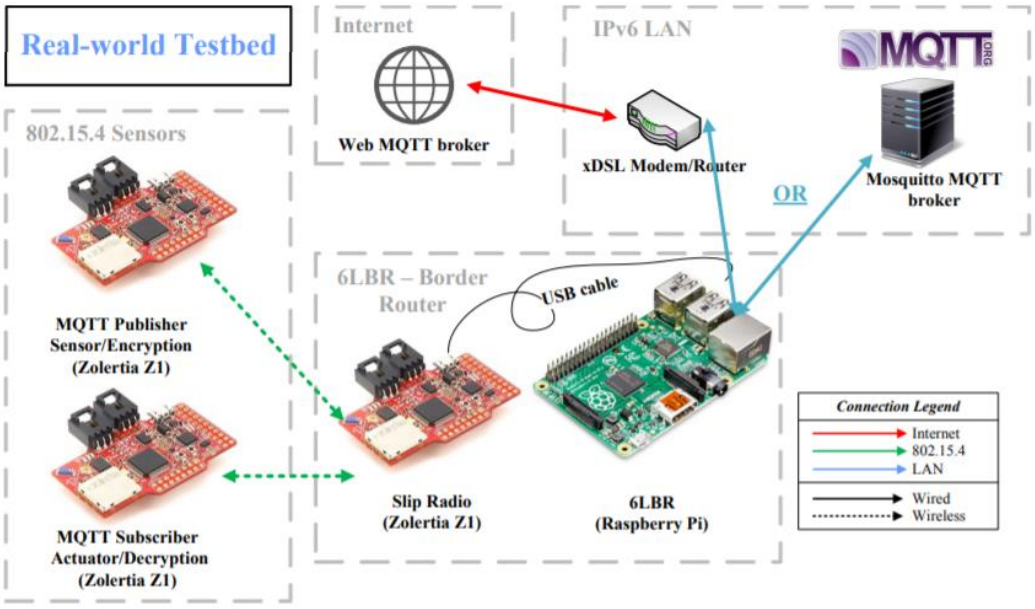


Figure 2.7: Testbed topology of the system [13].

Two different evaluation environments were configured and used during the phase: i) a virtual environment ii) a real-world environment. In the first, all the WSN motes programmed as MQTT clients were running inside the Cooja simulator on virtual Zolertia Z1 motes and were connected to a MQTT broker running on the VM's localhost with the use of Contiki's Native Border Router. On the second environment, the MQTT clients ran on real Zolertia Z1 motes connected to a MQTT broker running on a Windows machine connected to the LAN, with the use of 6lbr as Border Router that was configured on a Raspberry Pi. The principal arrangement was utilized during code improvement for testing and troubleshooting purposes, while the subsequent arrangement was utilized for assessing the usage on genuine stages. In the latter case, the motes were placed close to each other, to eliminate possible transmission problems that could affect the performance evaluation. Additionally, the relevance of the proposed arrangements is surveyed with regards to a genuine modern application, breaking down the system attributes and necessities of a real, working breeze park, as an agent use instance of modern systems [13].

2.3 Summary

In this Chapter, we have discussed about some topic related with MQTT, Development of Message Queuing Telemetry Transport (MQTT) based Vehicle Accident Notification System, Secure MQTT for Internet of Things (IoT), Correlation Analysis of MQTT Loss and Delay According to QoS Level, Performance Evaluation of MQTT and CoAP via a Common Middleware, Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things, Experimental and Environmental Control and Monitoring of the Chemistry Laboratory and Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol which all related works of development.

CHAPTER 3
METHODOLOGY

3.1 Introduction

MQTT is a machine-to-machine messaging protocol, designed to provide lightweight publish/subscribe communication to “Internet of Things” devices. It is commonly used for geo-tracking fleets of vehicles, home automation, environmental sensor networks, and utility-scale data collection.

It is also important to realise that when planning security implementation that one must consider the capabilities of your MQTT clients as well as your broker.

Internet of Things and MQTT both are took off now. Mosquitto is the first MQTT broker which is open source. It was created around 2008. And in 2014 it became the name of Eclipse Mosquitto project. The open source MQTT client libraries were published in 2012 as Paho project C, Python, JavaScript and Java. And since then it is growing day by day. The most remarkable things are the broker version 3.1.1 becomes OASIS standard in late 2014 and in 2016 MQTT become an ISO standard.

MQTT is a widely-used general purpose IoT application layer protocol, usable in both constrained and powerful devices, which coordinates data exchanges through a publish/subscribe approach. In this paper we propose a methodology to increase the security of the MQTT protocol, by including Usage Control in its operative workflow. The inclusion of Usage Control enables a fine-grained dynamic control of the rights of subscribers to access data and data-streams over time, by monitoring mutable attributes related to the subscriber, the environment or data itself. We will present the architecture and workflow of MQTT enhanced through Usage Control, also presenting a real implementation on Raspberry Pi 3b for performance evaluation

3.2System Architecture

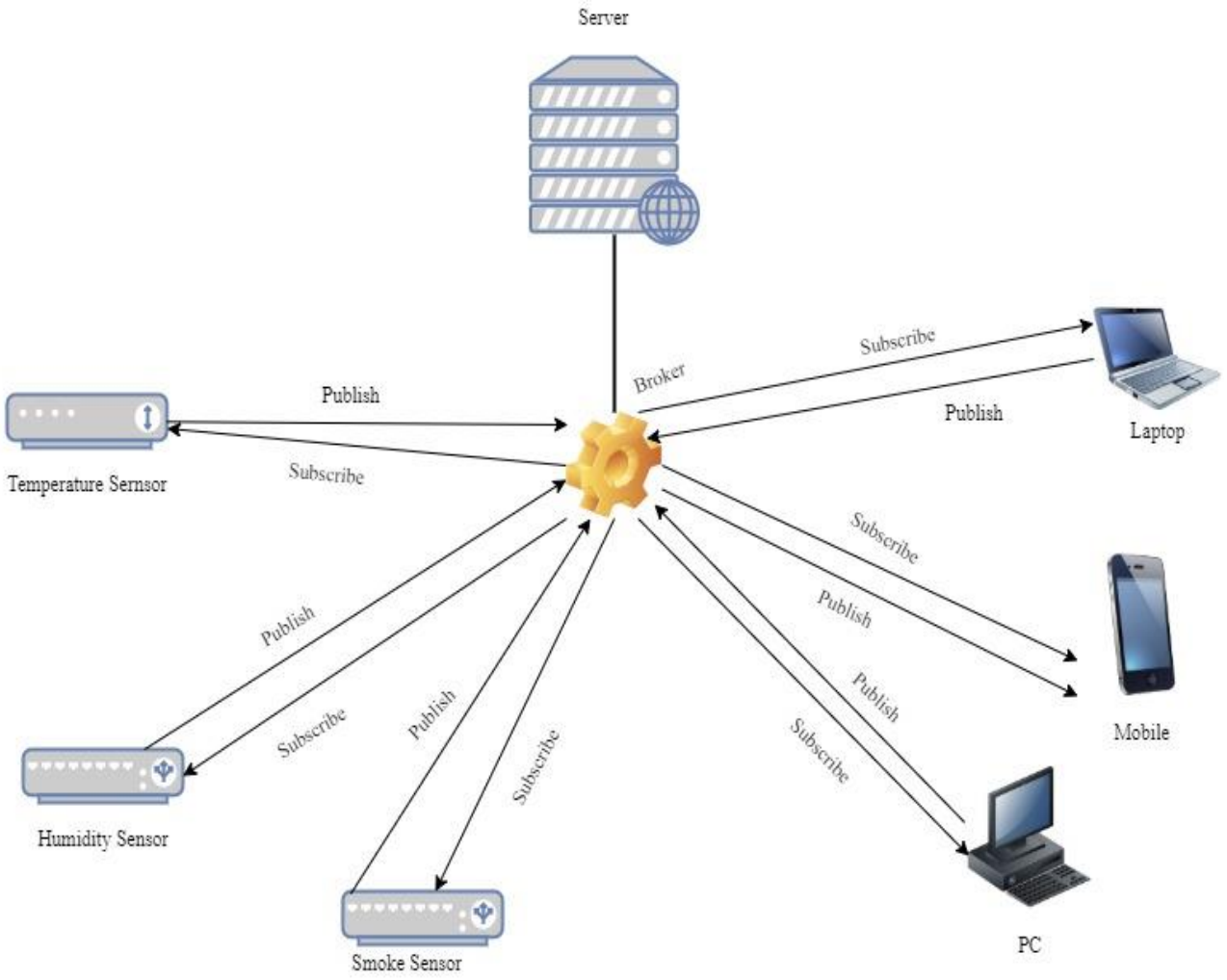


Figure 3.1: System Architecture

3.3 Check Broker

We are using Mosquitto MQTT broker as out broker. So, we must check that the mosquito service file is working well.

```
pi@raspberrypi:~ $ systemctl status mosquitto.service
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset
   Active: active (running) since Sat 2020-01-18 18:00:00 +06; 1 weeks 0 days ag
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
 Main PID: 520 (mosquitto)
    Tasks: 1 (limit: 2200)
   Memory: 2.2M
   CGroup: /system.slice/mosquitto.service
           └─520 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

Figure 3.2: Check Broker

3.4 WebSocket Connection Status

Check the connect() function of mqtt.js file, in this function there are number of variables which are liable for a successful connection suc as hostname, port etc.

```
function connect() {
    console.log("Connecting to server")
    var hostname = "192.168.0.111";
    var port = "9001";
    clientId = "attendanceWeb_" + makeid() + String(Date.now());
    var keepAlive = Number(10);
    var timeout = Number(3);
    client = new Paho.MQTT.Client(hostname, Number(port), clientId);
    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;
    var options = {
        invocationContext: {host: hostname, port: port, path: client.path, clientId:
        clientId},
        timeout: timeout,
        keepAliveInterval: keepAlive,
        useSSL: true,
        cleanSession: true,
        reconnect: true,
        onSuccess: onConnect,
        onFailure: onFail
    };

    client.connect(options);
```

Figure 3.3: Check Connection Status.

If the mqtt.js file is ok then check the WebSocket status of webpage

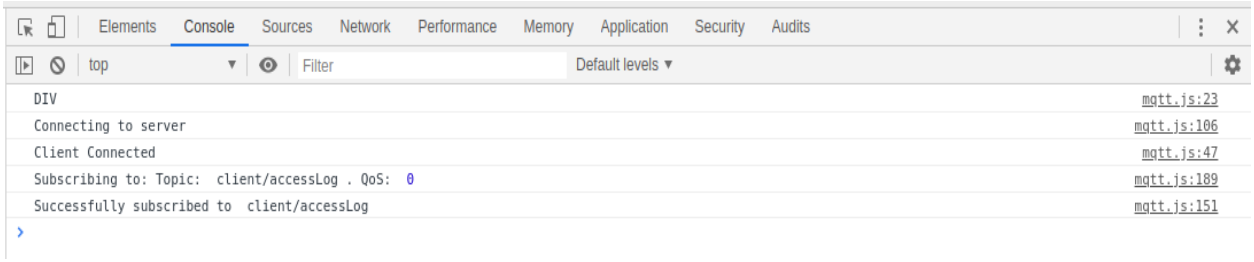


Figure 3.4: WebSocket Status.

3.5 Publish Message

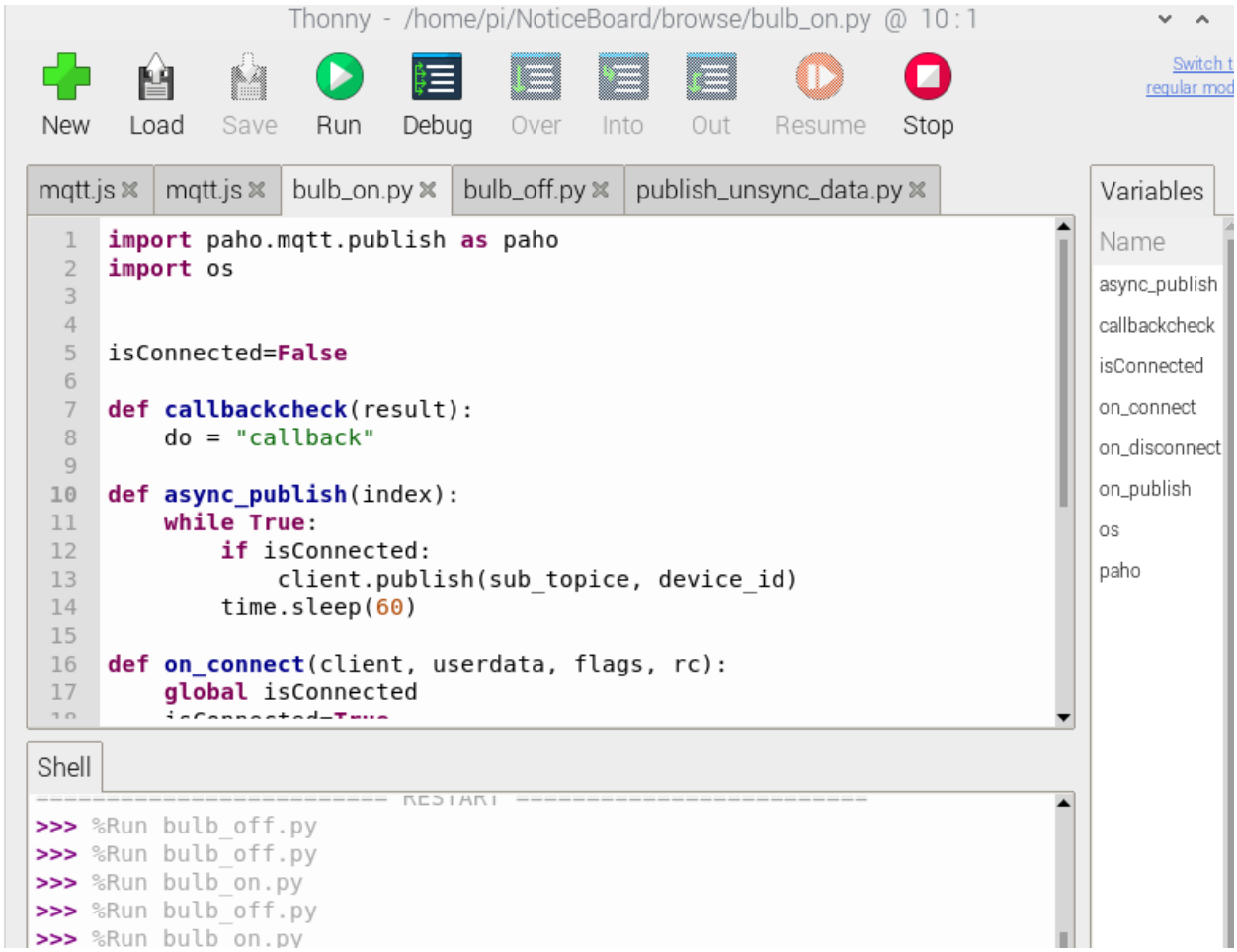


Figure 3.5: Message Publish.

3.6 Get Publish Data

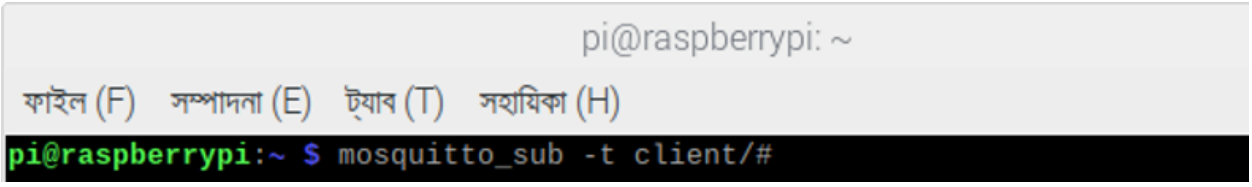


Figure 3.6: Get Message Publish

3.7 Secure Messaging

When it comes to IoT network security, there are three basic concepts to keep in mind: **identity**, **authentication** and **authorization**. In each MQTT scenario, there is a client and a broker. A client can be anything ranging from a microcontroller to a server. Anything that makes a connection to a broker is considered a client.

A broker receives all messages and coordinates the publishing of messages to clients that are subscribed. The broker is responsible for persisting connections, as well as identifying and authorizing the transfer of data to MQTT clients. The MQTT connection is only between one client and a broker.

3.7.1 Username and Password Approach

Using authentic username and password, the mosquitto MQTT broker can be configured as to require client authentication before broker connection established or permitted. A clear text username and password combination is not secure without transport encryption like ssl.

Firstly, we will add username and password authentication and next we will add some ssl certificate to secure the communication. To restrict access to a broker easily using username and password authentication is a good way.

The restrictions forms are client id, topic, qos, username/password etc which all are implemented in this MQTT broker.

After implementing all the credentials to broker, it is now task for client to add all these credentials to connect, publish and subscribe with maintaining all the restrictions.

For configuring the Mosquitto broker, we will need to do some tasks. The tasks we have to do are given below:

- Create/generate a file for password
- Change the mosquitto.conf file to force to use password

3.7.2 Create a password file

The mosquitto_passwd utility file, it is installed with the client tools during installation of the Mosquitto broker will need to use to create the password file.

Many ways to do this:

```
mosquitto_passwd -c passwordfile user
```

Here “secure” is the username. Later for authentication we need to this username in different purpose.

After that the terminal will ask for a password which will be assigned for the user we have specified.

We have to be careful to give the password cause it will not be showed to the terminal during typing the password.

After that we can use the following command

```
mosquitto_passwd -b passwordfile secure secureconnection
```

It will help to add others additional users to the file, which means we can use multiple users.

The figure below showing the process:

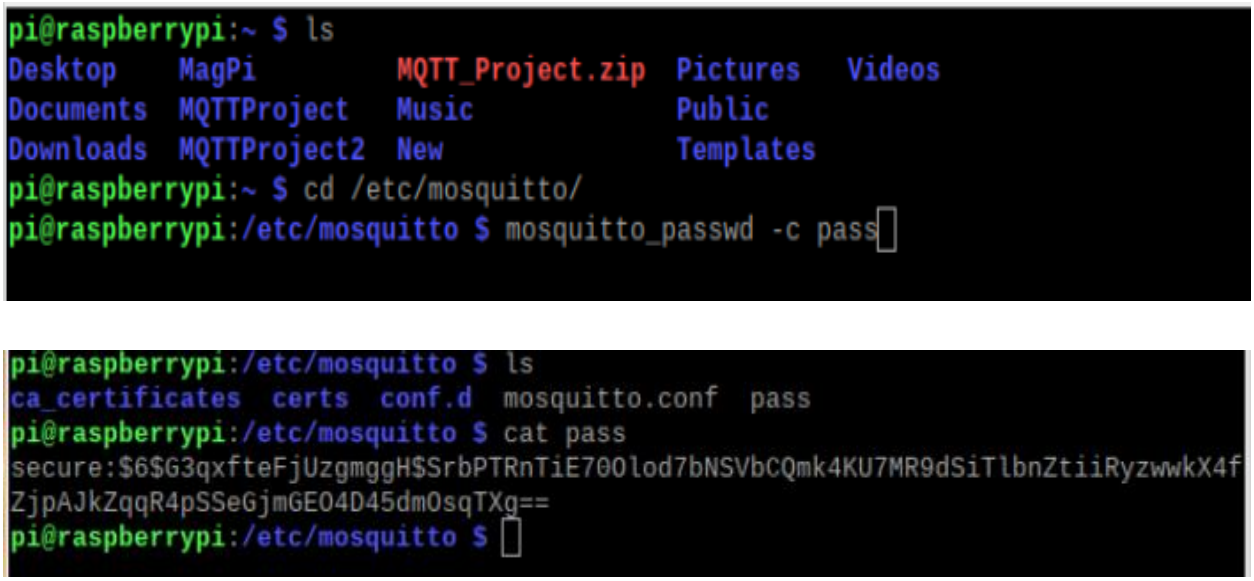


Figure 3.7: Mosquitto password.

We can also remove any users from our password configuration file using the following command.

```
mosquitto_passwd -D passwordfile secure
```

Above all the information are worked for Raspberry Pi.

3.7.3 Using the Password file

We have to copy password file into the directory etc/mosquitto folder in our Raspberry Pi and the we will change our mosquitto.conf file to use the password file we have copied.

There are two changes made. The changes made to mosquitto.conf file are setting the password file path and also setting allow anonymous to false cause by default it remains true.

Settings of mosquitto.conf

password_file /etc/mosquitto/pass.txt

allow_anonymous false

Password File Example

We have shown an example password file pass

The file has four users:

- secure
- pi
- user1
- user2

3.7.4 Reloading the Password File

Every time we change the mosquitto.conf file and the password file we need to restart the mosquitto broker.

In Raspberry Pi we can reload the file. Without restarting the broker we can reload the files by using following command:

kill-HUP 519 #Here 519 is process ID

The terminal is showing that the conf file is reloaded

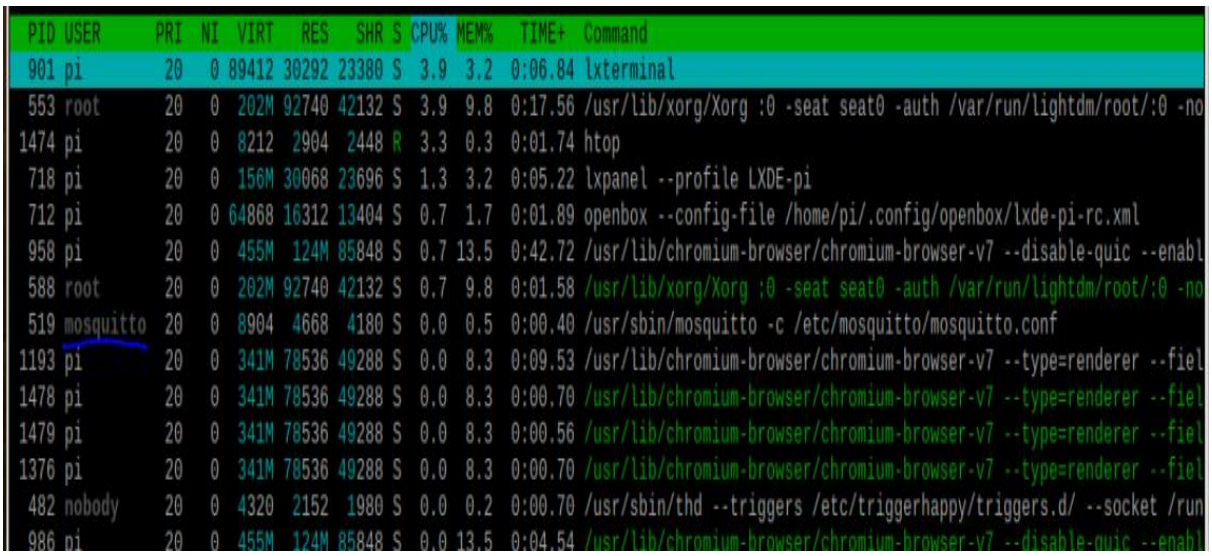


Figure 3.8: Console log status.

After successfully setting the username and password to MQTT broker set username and password to mqtt.js file for the authenticated connection of WebSocket.

```
function connect() {
    console.log("Connecting to server")
    var hostname = "192.168.0.111";
    var port = "9001";
    clientId = "attendanceWeb_" + makeid() + String(Date.now());
    var user = "secure";
    var passw = "secureconnection";
    var keepAlive = Number(10);
    var timeout = Number(3);
    client = new Paho.MQTT.Client(hostname, Number(port), clientId);
    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;
```

Figure 3.9: Authenticated connection of WebSocket.

3.8 SSL Approach

The Firstly we have to create a private key in this process. So, we are going to connect the Raspberry Pi using ssh. It can be connected with a remote desktop or directly with a monitor also. The most important thing is that we must install the OpenSSL first before doing this step on our Raspberry Pi. Through Openssl we will generate all the certificate files.

Secondly, we have to create a directory in where we will store all the certificates we will create and then we will move forward to create a private key.

```
openssl genrsa -out mosq-ca.key 2048
```

The command will create a 2048-bit key called mosq-ca.key. And the result of this command is showing the figure below:

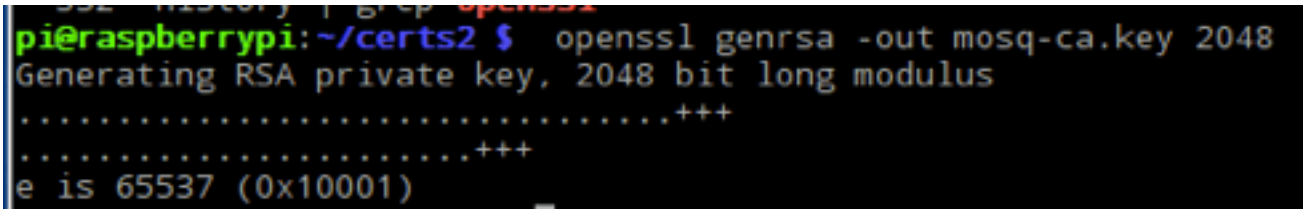


Figure 3.10: Creating 2048bit key.

After creating the private key, the next task is to create an X509 certificate which will use the private key that we have created before. Now opening another terminal in the same directory, we will store the private key by writing the following code:

```
openssl req -new -x509 -days365 -key mosq-ca.key -out mosq-ca.crt
```

After submitting the command, we have used the server will ask for some information for creating the certificate and we have provided these. The following picture shows the terminal after submitting the command:

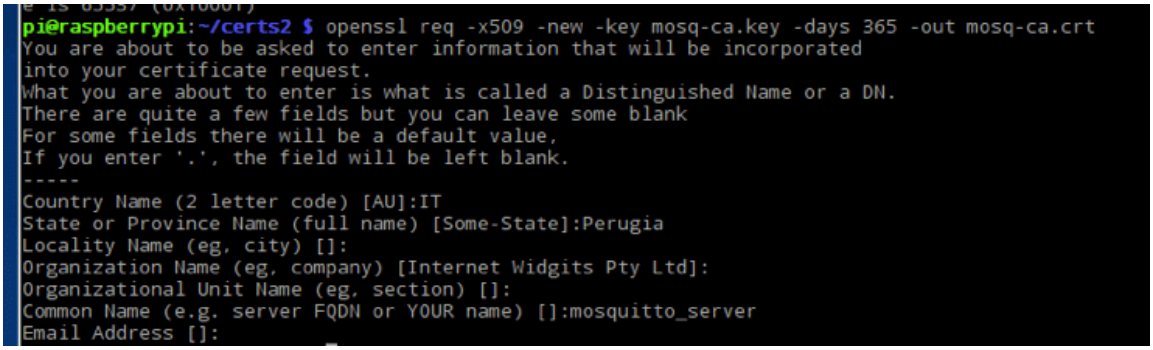


Figure 3.11: Key certificated.

3.8.1 Generating the MQTT Server Certificate

Now we can create MQTT server certificates and private key as our private key and the required certificates are ready to use.

To create private key:

```
openssl genrsa -out mosq-serv.key 2048
```

After that we can move on the creating server certificate. For this purpose, we will create a Certificate Signing Request (CSR). The certificate we have created would be sent to certification authority, they will verify it and after that they will return the certificate. Here we are using the self-signed certificate for avoiding the complexity:

```
openssl req -new -key mosq-serv.key -out mosq-serv.csr
```

So we have generated the certificates and all other things we need. Now we can create the all the credentials and certificates to use in our server:

```
openssl x509 -req -in mosq-serv.csr -CA mosq-ca.crt -CAkey mosq-ca.key -CAcreateseri
al -out mosq-serv.crt -days 365 -sha256
```

The whole procedure is completed to secure our MQTT server. Now the time to verify the cerficates and all others:

```
openssl x509 -in mosq-serv.crt -noout -textjavascript:void(0)
```

Now you should see the certificate.

3.9 How to Configure MQTT Mosquitto Server to Secure MQTT

Now the certificates are ready to configure the Mosquitto beoker. For configuring the MQTT Server we will use mainly three certificates. The list of certificates is given below:

- mosq-ca.crt
- mosq-serv.crt
- mosq-serv.key

Now we have to edit the mosquitto.conf file which holds all the configuration parameters. We will add some extra lines in this file. These lines are required for ensuring ssl certificate to the broker. The lines are given below:

```
listener 8883
cafile /home/pi/ssl-cert-mosq/mosq-ca.crt
certfile /home/pi/ssl-cert-mosq/mosq-serv.crt
keyfile /home/pi/ssl-cert-mosq/mosq-serv.key
```

The We have stored our certificates to the path /home/pi/ssl. We also have changed the default Mosquitto MQTT port to 8883.

Now will test that everything is running well. For that, firstly we have to stop the Mosquitto service and then restart it. After that it can work with the lines, we have added to mosquitto.conf file. The command to stop/start the service file is following:

```
sudo service mosquitto stop
sudo service mosquitto start
```

We have completed the steps for securing MQTT protocol. Our MQTT protocol is now secure and encrypted. For testing the certificates are working well, we will use a Java-based MQTT client from our windows operating system-based laptop.

The security testing of Mosquitto through SSL/TSL

Firstly, we will check the connection is correctly configured or not. We will verify it first because we have changed many connection parameters in the mosquitto.conf file. To ensure this we are using MQTT.fx which is a Java based MQTT client. We have to install it to our computer. And after that we have to change the settings and connection parameters of client and also have to create a profile with all the proper information as shown in the figure below:

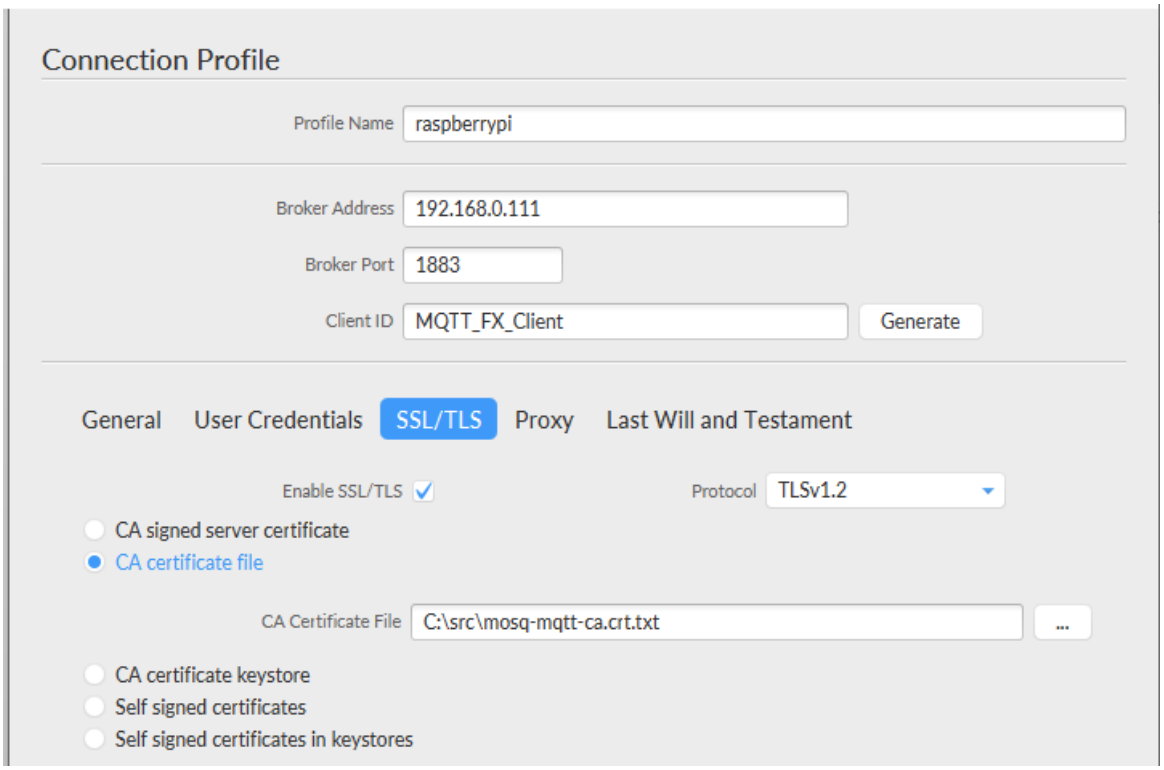


Figure 3.12: Configure MQTT Mosquitto Server to Secure MQTT.

We have already enabled the SSL/TSL configuration, by providing the certificate files like mosq-ca.crt in previous steps.

So, now we can test/connect to the Mosquitto server for MQTT:

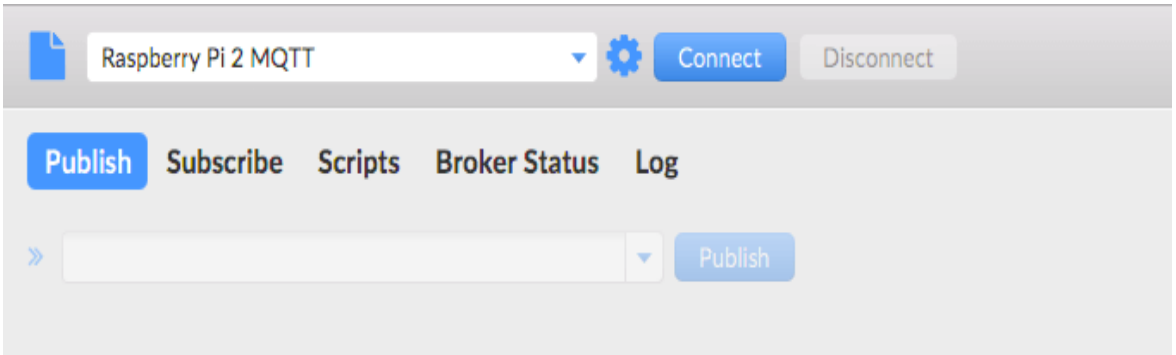


Figure 3.13: Configure Secure MQTT.

Click After pressing the connect button, the noticing thing is that the client will establish the connection the broker which can be checked to the log tab.

Now we have to check that our client gets the messages or not. For this purpose, we have to select the subscribe menu from the menu bar and then we have to choose the topic to which the client will subscribed.

On the other hand, in Raspberry Pi which contains the broker, we will send a message with same topic and same channel:

Before setup SSL certificate the WebSocket status given below

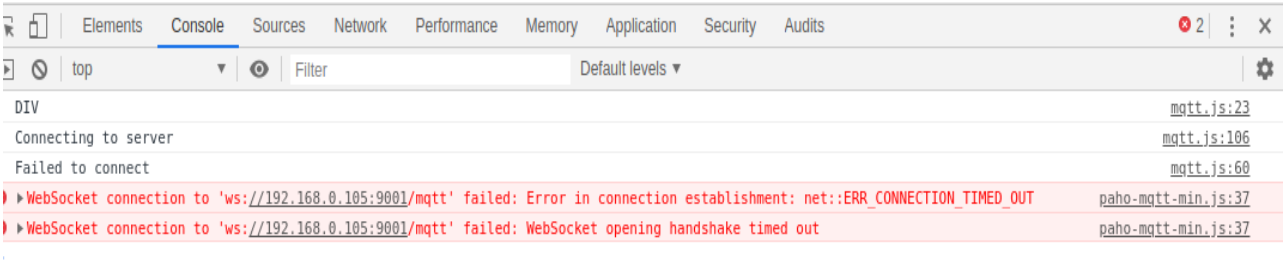


Figure 3.14: SSL WebSocket status.

After successfully setup SSL certificate to MQTT broker, we must change some security options of mqtt.js file for secure WebSocket connection with web page to broker

```
var options = {
    invocationContext: {host: hostname, port: port, path: client.path, clientId:
    clientId},
    timeout: timeout,
    keepAliveInterval: keepAlive,
    useSSL: true,
    cleanSession: true,
    reconnect: true,
    onSuccess: onConnect,
    onFailure: onFail
};
```

Figure 3.15: Secure WebSocket connection.

```
mosquitto_pub -p 8883 -t "test" -cafile mosq-mqtt-ca.crt -m "Hello MQTT" -d -h 192.168
.1.8
```

The Log table result for connection and disconnection status, which is also showing that the sending packet status.

```
pi@raspberrypi:~/certs2 $ mosquitto_pub -p 8883 --cafile mosq-ca.crt -t test -m "Hello MQTT" -d -h 192.168.1.8
Client mosqpub/2172-raspberryp sending CONNECT
Client mosqpub/2172-raspberryp received CONNACK
Client mosqpub/2172-raspberryp sending PUBLISH (d0, q0, r0, m1, 'test', ... (10 bytes))
Client mosqpub/2172-raspberryp sending DISCONNECT
```

Now time to check subscriber side. The subscriber side located in another computer. The status of the subscriber side given below:

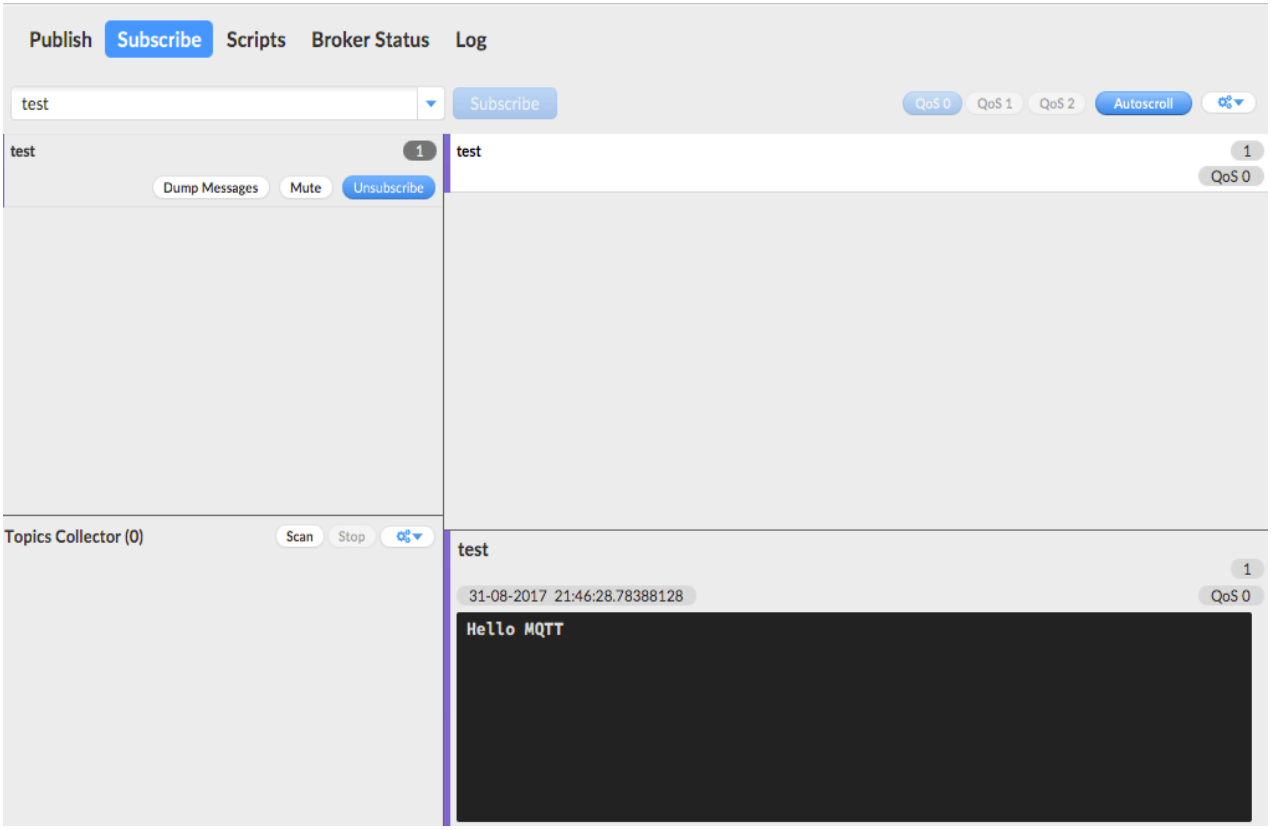


Figure 3.16: Subscriber side.

CHAPTER 4
RESULT & ANALYSIS

4.1 Experimental Setup

An important part of this project was ensuring that the system met certain time bounds. In order to test that these bounds were met, additional tests of the system were implemented. Since many such IoT devices would need to operate in a constrained environment, Chrome network analyzer was used to test the system in varying levels of packet loss and latency. In order to measure latency time from sending to receiving, the client node was modified to subscribe to its own updates, time them from publish to reception, and output the results to a web page. In this manner, a total roundtrip time could be measured from interrupt detection to reception of the state packet. By measuring the response time across varying network and system conditions, a map of system performance in constrained environments would be created. During latency tests, packet losses were additionally measured.

4.1.1 Hardware Platform

The full project program implemented in Raspberry Pi 3B. It uses a 1.2GHz 64-bit quad-core Arm Cortex-A53 CPU with 1GB RAM and the operating system is Raspbian.

4.1.2 Software Tools

In this research Python 3.4 as our main development platform and for mathematical analysis Numpy 1.11.1, scipy 0.18.0 are used and for MQTT publish/subscribe paho-mqtt 1.5.0 is used. The graphical display is done by Chromium, which was in raspbian os by default. There are some primary toolkits for WebSocket in browser. paho-mqtt.js and mqtt.js are some examples. As mqtt broker mosquitto has installed in raspberry pi.

4.2 Result and Analysis

Check all the file are loaded successfully, where the loading time is an important factor. The figure shows that all the dependencies take very short time. The time can be lessened more if the server processing power increased.

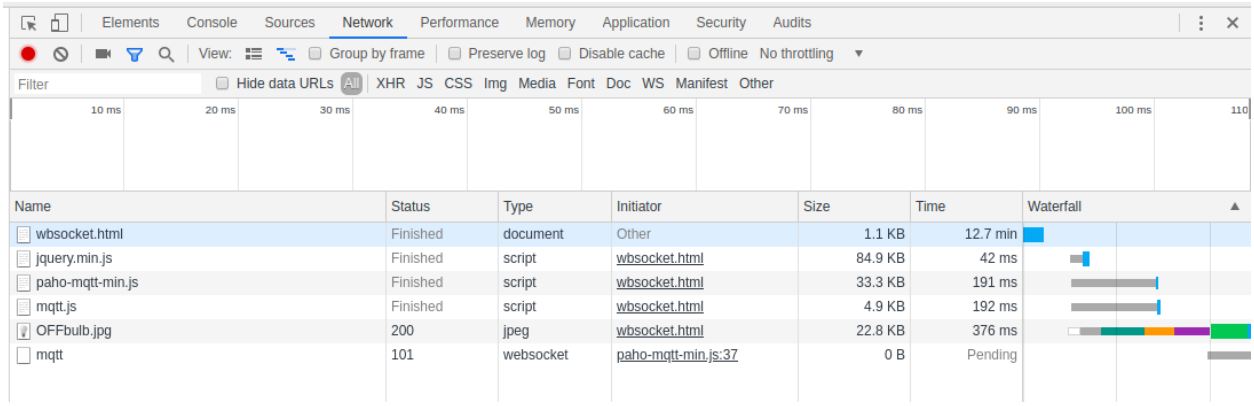


Figure 4.1: Server processing power.

Next approach is testing MQTT connection over port 1883. When there is no authentication set, anyone from any device can connect to this device. But after setting authentication and ssl certificate, it becomes difficult to connect to MQTT broker without knowing the dependencies. After trying from other devices without proper dependencies, we get this message give in Figure

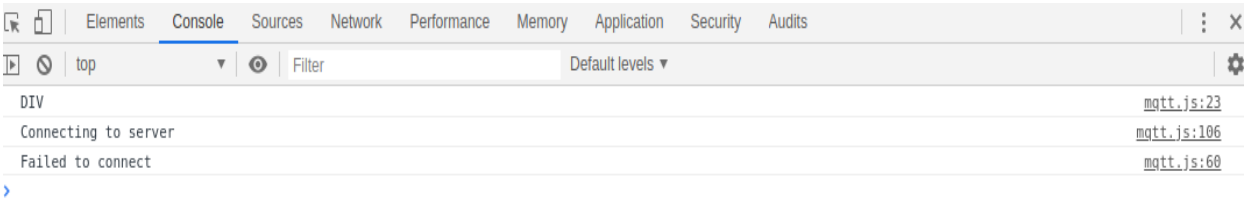


Figure 4.2: MQTT broker without dependencies.

So, in the output page, there is no change in bulb. The figure shows the websocket.html, and there is no change in bulb output.

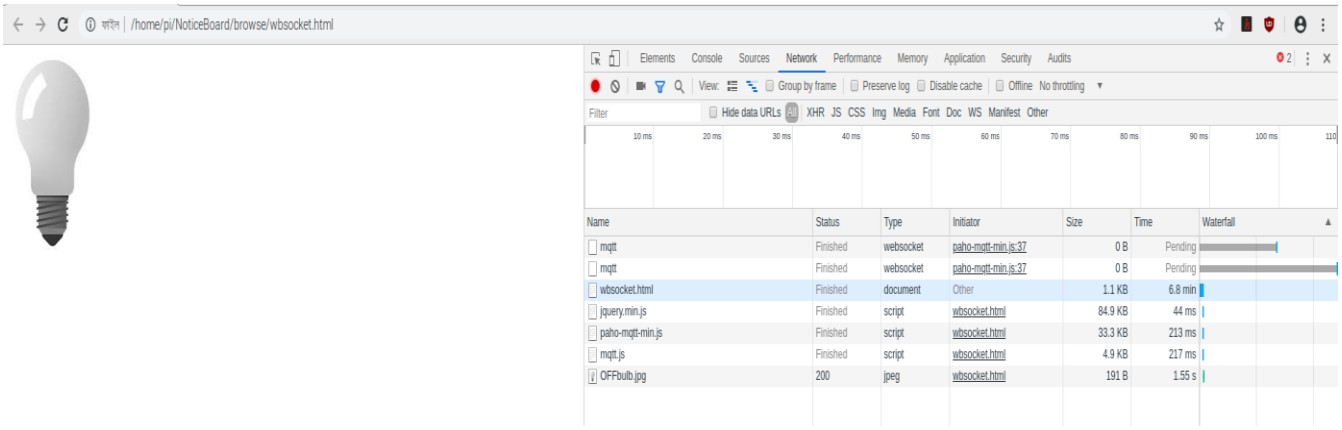


Figure 4.3: Add authentication to secure MQTT broker.

Now time to add authentication, we have shown before how to add authentication to secure MQTT broker. After adding all the dependencies and ssl certificate, we publish a message to on the bulb. The result is given below.

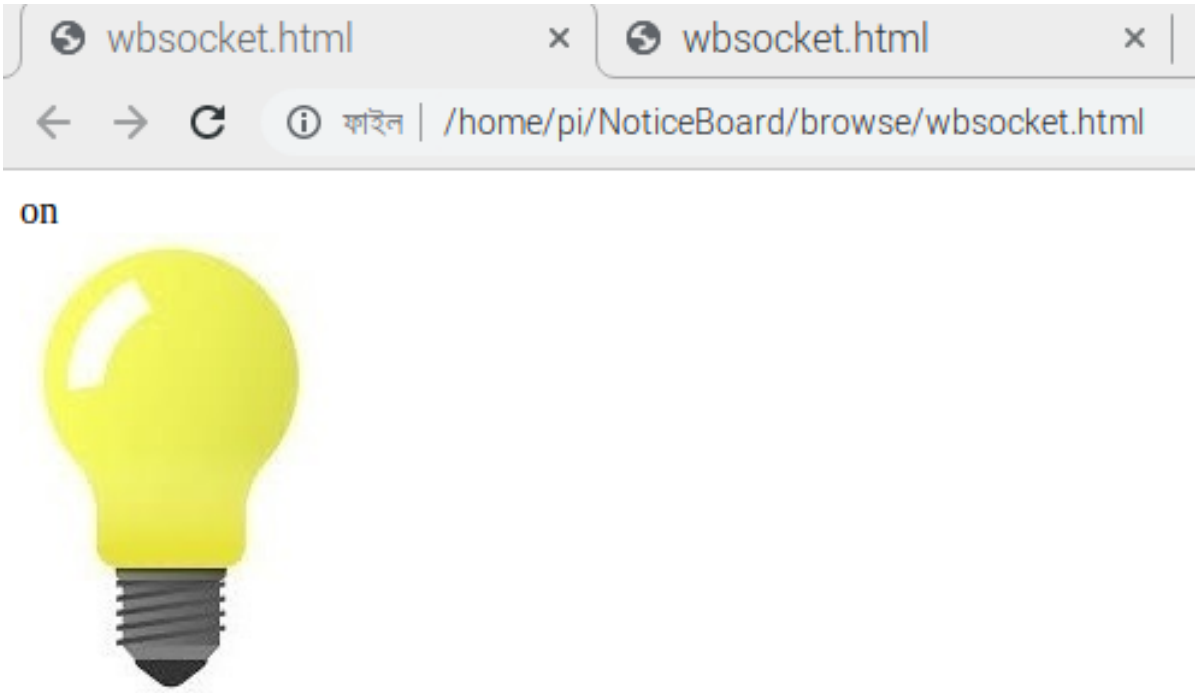
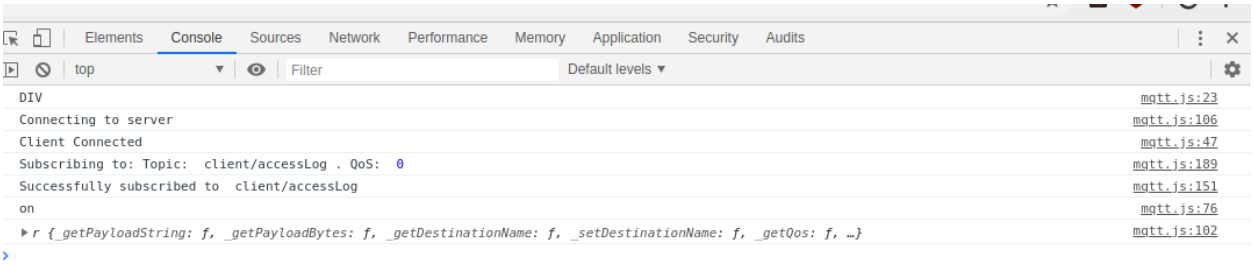


Figure 4.4: Message Publication.

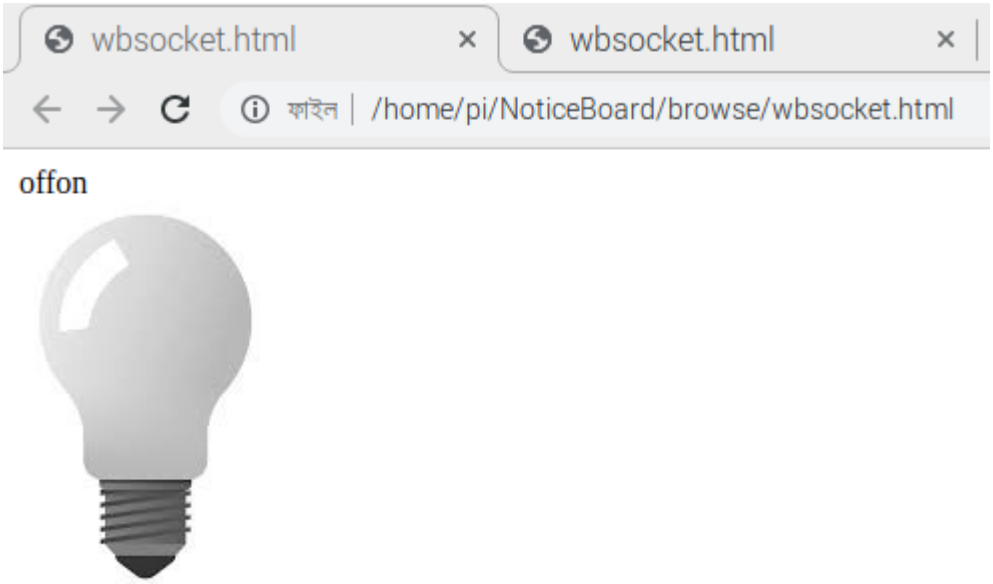
After successfully connected with broker, the WebSocket status is given to Figure where message publish for “on” the device.

The console showing that the WebSocket successfully connected to topic: client/accessLog.

It gets a message “on”, depending on this message it changes the bulb status.



After publishing “off” message the web server bulb status shown below:



WebSocket gets the published message:

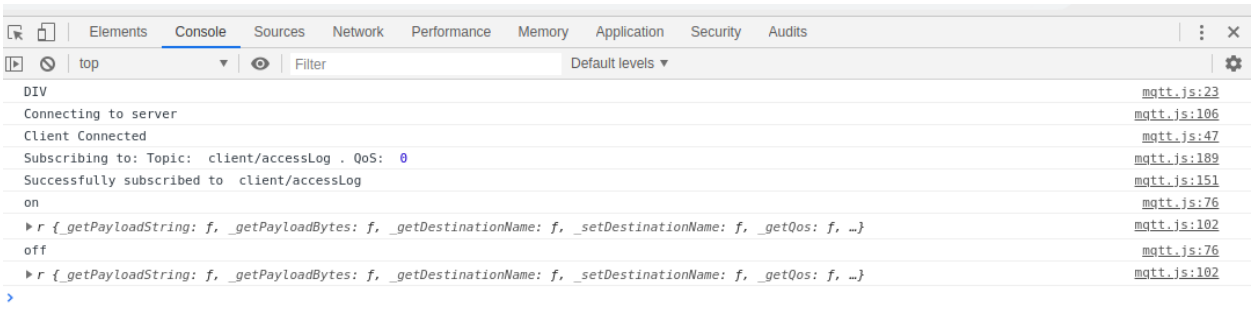


Figure 4.5: Broker Status.

The subscriber on server gets the messages which are printed in a terminal. This terminal is subscribing the same topic.

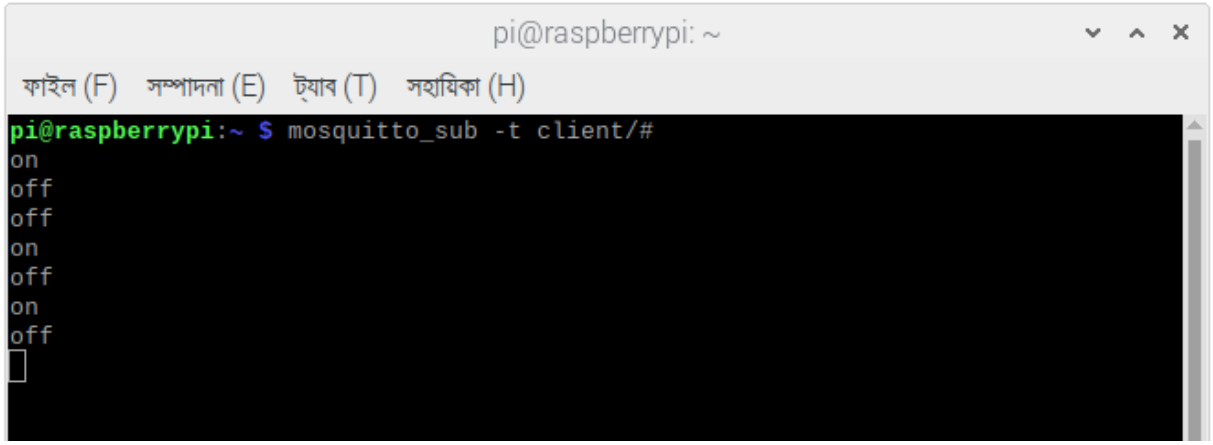


Figure 4.6: Server message.

4.3 Comparison between MQTT and HTTP

MQTT	HTTP
Message Queue Telemetry Transport.	Hyper Text Transfer Protocol.

In light data communication cases it takes 0.2 ms.	In light data communication cases it takes 20-30 ms.
Works on the publish and subscribe model to ensure efficient communication across platforms.	HTTP work well as its compact transmissions and low overhead will certainly lessen the strain on hardware in terms of memory and battery consumption.
If project is to let the fridge to communicate with the thermometer to adapt the engine pump, you can use the MQTT easily	If need to collect big data from around the world, then you can think to use HTTP

4.4 MQTT & HTTP response time

No. of property fields in the message	Avg. response time(ms) MQTT	Avg. response time (ms) HTTP
1	0.22	20
10	1	40
100	4	50

CHAPTER 5
CONCLUSION

5.1 Contribution

In this work, we have proposed a secured automated messaging system using Message Queuing Telemetry Transport that takes message from other devices such as sensor, mobile, laptop and accept only the ones with proper authenticity. This work specifically demonstrated an application of the system can successfully communicate with IOT support devices without any data loss and also secured.

An experimental framework has been developed using a broker, some python and JavaScript scripts with associated supportive software tools to evaluate the performance of the proposal system. In this work one hypothetical webpage were used as sample output device. The performance of the system was then evaluated in term of accuracy, time and security of communication system.

5.2 Future Work

We would like to think the internet was always ready for e-commerce and that the cryptography and security of systems protecting it have always been fit for purpose. However, the truth is that our understanding of secure systems and our expectations have evolved over time. SSL Version 1 never made it outside of Netscape and SSH version 1 is considered badly broken. We have iterated these protocols over the last several decades to arrive at today's level of transport security.

Security in personal messaging has likewise evolved from obscure, niche origins (such as OTR over what was originally called Jabber). The first attempt to bring end-to-end security to this space was TextSecure and RedPhone around 2012, a time in which all other messaging applications relied only on transport security. These applications were

merged into Signal. Following the success of Signal, the protocol was licensed by and integrated into WhatsApp – among others – and is now used daily by around 1.5 billion people. Alternative products such as Wire, Telegram, and Facebook Messenger either launched with similar security standards, or rolled them out on their platforms.

Teserakt's vision is that the same evolution is happening in the embedded, IoT, V2X, and M2M spaces. Today, messaging protocols such as MQTT and Kafka are at best providing transport security. We want the state of the art of end-to-end-security to apply to the smart home, to critical national infrastructure, to the cars you drive, and to the medical devices you use.

There is now a keen focus on updating and extending the functionality of IoT devices, similar to what we've all become accustomed to with the modern-day world of mobile devices and the smartphone. In this approach, companies can create and publish new applications and services via their own branded IoT app stores and extend device lifecycles as well as increase customer retention and revenues.

The app store approach also encourages the creation of license models and revenue streams based on specific feature enablement and user behavior. For example, Tesla can remotely configure their cars to enable self-driving capabilities in different models. Likewise, IoT devices can allow for mass customization of devices based on specific customer needs, licensing, and market demands for a device manufacturer's brand-specific offering.

Devices equipped with the ability to properly support revisions and updates of applications typically have lower support costs, too. The fact that applications can automatically update to new operating-system or application versions means businesses can be assured that all of their users are on the latest and supported version. Rollback features can also give hardware components such as webcams, security cameras, and other connected devices an added layer of security, in case the hardware is ever compromised through improper software distribution. The previous high-profile

exposures of security attacks such as Meltdown and Spectra show that, unfortunately, there's no magic bullet to security.

The damage and disruption that could be wrought by an attack of this nature to our increasingly connected homes would be substantial. As people rely ever more on connected-home devices, the downtime of our appliances, like heating, water, or refrigeration, needs to be minimal. Therefore, an attack response must be able to keep systems operational as they move through a stream of software updates to protect against any unwelcomed threats.

It's no longer the case that you can write software once and expect it to be secure and bug-free forever. Software will fail. The key is how quickly and comprehensively a business can respond to that failure—it's a true business differentiator.

REFERENCES

[1] S. Behnel, L. Fiege, G. Muehl, “On Quality of Service and Publish Subscribe,” Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops, July, 2006.

[2] Muneer Bani Yassein, Mohammed Q. Shatnawi. “Internet of Things: Survey and open issues of MQTT Protocol.” ICEMIS2017, Monastir. IEEE, 2017

[3] Upadhyay, Yuvraj, Amol Borole, and D. Dileepan. "MQTT based secured home automation system." Colossal Data Analysis and Networking (CDAN), Symposium on. IEEE, 2016.

[4] T. Mahmoodi, V. Kulkarni, W. Kellerer, P. Mangan, F. Zeiger, S. Spirou, I. Askoxylakis, X. Vilajosana, H. J. Einsiedler, and J. Quittek, “VirtuWind: virtual and programmable industrial network prototype 2017 IEEE Symposium on Computers and Communications (ISCC) deployed in operational wind park,” Trans. Emerg. Telecommun. Technol., vol. 27, no. 9, pp. 1281–1288, 2016.

[5] D. Evans, “The Internet of Things: How the Next Evolution of the Internet Is Changing Everything”, Cisco, April 2011.

[6] S. C. Mukhopadhyay, “Internet of things: challenges and opportunities”, vol. 9.; 9, pp. 1{7. Springer, 2014.

[7] Bharat Naresh Bansal, Vivek Garg.” Development of Message Queuing Telemetry Transport (MQTT) based Vehicle Accident Notification System.” International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-9 Issue-2, December, 2019.

[8] Meena Singh, Rajan MA, Shivraj VL, and Balamuralidhar P, “Secure MQTT for Internet of Things (IoT)”. 2015 Fifth International Conference on Communication Systems and Network Technologies.

[9] S. Lee, H. Kim, D. K. Hong, and H. Ju, “Correlation analysis of MQTT loss and delay according to QoS level,” in *International Conference on Information Networking*, 2013, pp. 714–717.

[10] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, “Performance evaluation of MQTT and CoAP via a common middleware,” in *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*, 2014.

[11] Y. F. Gomes, D. F. S. Santos, H. O. Almeida, and A. Perkusich, “Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things,” in *2015 IEEE International Conference on Consumer Electronics, ICCE 2015*, 2015, pp. 200–201.

[12] S. Wilson and J. Frey, “The Smart Lab: Experimental and environmental control and monitoring of the chemistry Laboratory,” in *2009 International Symposium on Collaborative Technologies and Systems, CTS 2009*, 2009, pp. 85–90.

[13] S. Katsikeas *et al.*, “Lightweight & secure industrial IoT communications via the MQ telemetry transport protocol,” in *Proceedings - IEEE Symposium on Computers and Communications*, 2017, vol. 0, pp. 1193–1200.

APPENDIX

CODE SEGMENT

```
// Create a client instance
client = null;
connected = false;
subscribed = false;

/*
#####################################
#     subscription code start      #
#####################################
*/

var previous_topic;
var sub_topic;
var messageReplace;
var html_element_id;
var html_element;
var tag;

function stellar_mqtt(operation, html_id, client_id, replace) {
    messageReplace = replace || false;
    html_element = html_id;
    tag = document.getElementById(html_element.substr(1)).tagName;
    console.log(tag);
    sub_topic = client_id + "/" + operation;
    if (connected == false) {
        connect();
```

```
    }
    else {
if (previous_topic == undefined) {
subscribe(sub_topic)
}
else {
unsubscribe(previous_topic);
subscribe(sub_topic);
}
}
}


/*
#####################################
#      subscription code END       #
#####################################
*/

function onConnect(context) {
console.log("Client Connected");
connected = true;
if (previous_topic != undefined  && previous_topic != sub_topic) {
unsubscribe(previous_topic);
}
if (sub_topic != undefined) {
subscribe(sub_topic);
}
$('#connectionStatusMQTT').change();
}
```

```
function onFail(context) {
    console.log("Failed to connect");
    connected = false;
    $('#connectionStatusMQTT').change();
}


function onConnectionLost(responseObject) {
    if (responseObject.errorCode !== 0) {
        console.log("Connection Lost: " + responseObject.errorMessage);
    }
    subscribed = false;
    connected = false;
}


function onMessageArrived(message) {
console.log(message.payloadString);
var image = document.getElementById('Image');
if (message.payloadString=="on"){
image.src = "https://media.geeksforgeeks.org/wp-content/uploads/ONbulb.jpg";
}
else{
image.src = "https://media.geeksforgeeks.org/wp-content/uploads/OFFbulb.jpg";
}
if (messageReplace == true) {
if (tag == "INPUT") {
$(html_element).val(message.payloadString);
```

```
$(html_element).change();
}
else {
$(html_element).text(message.payloadString);
}
}
else {
if (tag == "INPUT") {
$(html_element).val(message.payloadString + $(html_element).val());
$(html_element).change();
}
else {
$(html_element).prepend("<span id='stellar_data'>" + message.payloadString +
"</span>");
}
}
console.log(message);
}

function connect() {
    console.log("Connecting to server")
    var hostname = "192.168.0.111";
    var port = "9001";
    clientId = "attendanceWeb_" + makeid() + String(Date.now());
    var user = "secure";
    var passw = "secureconnection";
    var keepAlive = Number(10);
    var timeout = Number(3);
    client = new Paho.MQTT.Client(hostname, Number(port), clientId);
```

```
    client.onConnectionLost = onConnectionLost;
    client.onMessageArrived = onMessageArrived;
    var options = {
        invocationContext: {host: hostname, port: port, path: client.path, clientId: clientId},
        timeout: timeout,
        keepAliveInterval: keepAlive,
        useSSL: true,
        cleanSession: true,
        reconnect: true,
        onSuccess: onConnect,
        onFailure: onFail
    };

    client.connect(options);
}

function disconnect() {
    console.info('Disconnecting from Server');
    client.disconnect();
    connected = false;
}



/*
############ Un/subscribe Success Text function #############
*/

function unsubscribeSuccess(context) {
    console.info('Successfully unsubscribed from ', context.invocationContext.topic);
    subscribed = false;
```

```
}


function subscribeSuccess(context) {
    console.info('Successfully subscribed to ', context.invocationContext.topic);
    subscribed = true;
    previous_topic = sub_topic;
}

/*
############ Un/subscribe Failure Text function ##############
*/

function unsubscribeFailure(context) {
    console.info('Failed to  unsubscribe from ', context.invocationContext.topic);
}

function subscribeFailure(context) {
    console.info('Failed to subscribe in ', context.invocationContext.topic);
}

/*
############ Random Key Generate function ##############
*/

function makeid() {
    var text = "";
    var possible =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
```

```
    for (var i = 0; i < 8; i++)
        text += possible.charAt(Math.floor(Math.random() * possible.length));
    return text;
}


function subscribe(sub_topic) {
qos = Number(0)
client.subscribe(sub_topic, {
onSuccess: subscribeSuccess,
onFailure: subscribeFailure,
invocationContext: {topic: sub_topic, qos: qos}
});
console.info('Subscribing to: Topic: ', sub_topic, '. QoS: ', qos);
}

function unsubscribe(previous_topic) {
client.unsubscribe(previous_topic, {
onSuccess: unsubscribeSuccess,
onFailure: unsubscribeFailure,
invocationContext: {topic: previous_topic}
});
}



<!doctype html>
<html>
<head>
```

```
</head>
<body>


<div id="user_signature"></div>

<img id="Image" onclick="turnOnOff()" src="https://media.geeksforgeeks.org/wp-content/uploads/OFFbulb.jpg">

<script src="js/jquery.min.js"></script>
<!-- EndInclude -->

<script src="js/paho-mqtt-min.js"></script>
<script src="js/mqtt.js"></script>

<!-- Configuration -->
<script>
var operation = "accessLog";
var html_id = "#user_signature";
var client_id = "client";
stellar_mqtt(operation, html_id, client_id);
</script>
<script>
$('#user_signature').change(function(){
console.log("changed");
var msg = $(this).val();
console.log(msg);
if (msg == "on"){
image.src = "https://media.geeksforgeeks.org/wp-content/uploads/OFFbulb.jpg";
}
```

```
else{
image.src = "https://media.geeksforgeeks.org/wp-content/uploads/ONbulb.jpg";
}
});
</script>
</body>
</html>


import paho.mqtt.publish as paho
import os

#paho.single("client/accessLog", "on")

isConnected=False

def callbackcheck(result):
    do = "callback"

def async_publish(index):
    while True:
        if isConnected:
            client.publish(sub_topice, device_id)
        time.sleep(60)

def on_connect(client, userdata, flags, rc):
    global isConnected
    isConnected=True

def on_disconnect(client, userdata, rc):
```

```
    global isConnected
    isConnected = False

def on_publish(client, userdata, result):
    do = "on publish"

try:
    #paho.single("client/accessLog",
"on",retained=False,auth={'username'="secure",'password'="secureconnection"},
hostname="192.168.0.116")
    #paho.single("client/accessLog", "on",retained=False, hostname="192.168.0.111")
    os.system("mosquitto_pub -t client/accessLog -m on")
    #os.system("mosquitto_pub -t client/accessLog -u secure -P secureconnectio -m on")
except:
    pass


import paho.mqtt.publish as paho
import os


isConnected=False

def callbackcheck(result):
    do = "callback"

def async_publish(index):
    while True:
        if isConnected:
            client.publish(sub_topice, device_id)
```

```
        time.sleep(60)

def on_connect(client, userdata, flags, rc):
    global isConnected
    isConnected=True

def on_disconnect(client, userdata, rc):
    global isConnected
    isConnected = False

def on_publish(client, userdata, result):
    do = "on publish"

try:
    #paho.single("client/accessLog",
"on",retained=False,auth={'username'="secure",'password'="secureconnection"},
hostname="192.168.0.116")
    #paho.single("client/accessLog", "on",retained=False, hostname="192.168.0.111")
    os.system("mosquitto_pub -t client/accessLog -m off")
    #os.system("mosquitto_pub -t client/accessLog -u secure -P secureconnectio -m on")
except:
    pass
```

```
var
previous_topic,sub_topic,messageReplace,html_element_id,html_element,tag;function
stellar_mqtt(e,n,o,t){messageReplace=t||!1,html_element=n,tag=document.getElementBy
Id(html_element.substr(1)).tagName,console.log(tag),sub_topic=o+"/"+e,0==connected?
connect():null==previous_topic?subscribe(sub_topic):(unsubscribe(previous_topic),subsc
```

ribe(sub_topic))}function onConnect(e){console.log("Client
Connected"),connected=!0,null!=previous_topic&&previous_topic!=sub_topic&&unsubs
cribe(previous_topic),null!=sub_topic&&subscribe(sub_topic),$("#connectionStatusMQ
TT").change()}function onFail(e){console.log("Failed to
connect"),connected=!1,$("#connectionStatusMQTT").change()}function
onConnectionLost(e){0!==e.errorCode&&console.log("Connection Lost:
"+e.errorMessage),subscribed=!1,connected=!1}function
onMessageArrived(e){console.log(e.payloadString);var
n=document.getElementById("Image");"on"==e.payloadString?n.src="https://media.gee
ksforgeeks.org/wp-
content/uploads/ONbulb.jpg":n.src="https://media.geeksforgeeks.org/wp-
content/uploads/OFFbulb.jpg",1==messageReplace?"INPUT"==tag?($(html_element).va
l(e.payloadString),$(html_element).change()):$(html_element).text(e.payloadString):"IN
PUT"==tag?($(html_element).val(e.payloadString+$(html_element).val()),$(html_eleme
nt).change()):$(html_element).prepend("<span
id='stellar_data'>"+e.payloadString+""),console.log(e)}function
connect(){console.log("Connecting to
server");clientId="attendanceWeb_"+makeid()+String(Date.now());var
e=Number(10),n=Number(3);client=new
Paho.MQTT.Client("192.168.0.111",Number("9001"),clientId),client.onConnectionLost=
onConnectionLost,client.onMessageArrived=onMessageArrived;var
o={invocationContext:{host:"192.168.0.111",port:"9001",path:client.path,clientId:clientI
d},timeout:n,keepAliveInterval:e,useSSL:!0,cleanSession:!0,reconnect:!0,onSuccess:onC
onnect,onFailure:onFail};client.connect(o)}function
disconnect(){console.info("Disconnecting from
Server"),client.disconnect(),connected=!1}function
unsubscribeSuccess(e){console.info("Successfully unsubscribed from
",e.invocationContext.topic),subscribed=!1}function
subscribeSuccess(e){console.info("Successfully subscribed to
",e.invocationContext.topic),subscribed=!0,previous_topic=sub_topic}function

unsubscribeFailure(e){console.info("Failed to unsubscribe from ",e.invocationContext.topic)}function subscribeFailure(e){console.info("Failed to subscribe in ",e.invocationContext.topic)}function makeid(){for(var e="",n="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",o=0;o<8;o++)e+=n.charAt(Math.floor(Math.random()*n.length));return e}function subscribe(e){qos=Number(0),client.subscribe(e,{onSuccess:subscribeSuccess,onFailure:subscribeFailure,invocationContext:{topic:e,qos:qos}}),console.info("Subscribing to: Topic: ",e,". QoS: ",qos)}function unsubscribe(e){client.unsubscribe(e,{onSuccess:unsubscribeSuccess,onFailure:unsubscribeFailure,invocationContext:{topic:e}})}client=null,connected=!1,subscribed=!1;